EECS 776 - Homework 5

Author: Jace Kline

Source Code

```haskell
import Data.List (nub)
import Data.Char (isUpper, isLower, toLower)
import System.IO
import System.Environment

{-
The sort hierarchy of the Linux 'sort' command is as follows:
    - lines that start with numbers are 'less than' lines that start with characters
        (already Haskell String semantics)
    - if same first character, then lowercase version string < uppercase version string
        (Created a SortStr data type to achieve this functionality)
    - otherwise, standard alphabetical order
-}

newtype SortStr = SortStr { toString :: String } deriving (Eq, Read)

instance Show SortStr where
    show (SortStr s) = show s

instance Ord SortStr where
    (SortStr s1) <= (SortStr s2) = case s1 of
        []     -> True
        (x:xs) -> case s2 of
            []     -> False
            (y:ys) -> if (toLower x == toLower y)
                      then case (isLower x, isLower y) of
                        (True, False) -> True
                        (False, True) -> False
                        _             -> def
                      else def
                        where def = (map toLower (x:xs)) <= (map toLower (y:ys))

{-
    The Option data type represents the options that can be applied when running the program
        -r option --> Reverse (reverses the sort order)
        -n option --> Numeric (attempts to parse the first column of file as numbers, then sorts)
        -u option --> Unique  (removes duplicate lines)
        -c option --> Compare (outputs a report that tells which line the original file became unordered at)
-}

data Option = Reverse | Numeric | Unique | Compare
    deriving (Eq, Show)

main :: IO ()
main = do
    args <- getArgs
    let (options, files) = parseInput args
    case (options, files) of
        (Left msg, _)       -> putStrLn msg
        (Right opts, files) -> sortCommand opts files

sortCommand :: [Option] -> [FilePath] -> IO ()
sortCommand opts files = do
    input <- case files of
                [] -> getContents
                fs -> readFiles fs
    let ls = nonemptys $ lines input
    let sorted = (if Numeric `elem` opts then sortNumeric else sortAlphabetic) ls
    if Compare `elem` opts
    then putStrLn $ compareOutput ls sorted
    else sequence_ $ map putStrLn $ optsApplied sorted
```

```
    where
        nonemptys = filter (\xs -> not (null xs))
        compareOutput ls sorted = let comp = compareLines ls sorted
                                  in case comp of
                                       Nothing -> "sort: file already sorted"
                                       Just n  -> "sort: disorder at line " ++ show n ++ ": " ++ (ls !! (n - 1))
        optsApplied sorted = ((if Unique `elem` opts then nub else id) . (if Reverse `elem` opts then reverse else id))
sorted

readFiles :: [FilePath] -> IO String
readFiles [] = return ""
readFiles (f:fs) = do
    this <- readFile f
    rest <- readFiles fs
    return $ this ++ "\n" ++ rest

compareLines :: [String] -> [String] -> Maybe Int
compareLines xs ys = go 1 xs ys
    where go :: Int -> [String] -> [String] -> Maybe Int
          go i [] _          = Nothing
          go i _ []          = Nothing
          go i (x:xs) (y:ys) = if x == y
                               then go (i+1) xs ys >>= \p -> return p
                               else return i

sortNumeric :: [String] -> [String]
sortNumeric xs = let keyified = [(((read :: String -> Double) . head . words) x, x) | x <- xs] :: [(Double, String)]
                 in map snd $ sort keyified

sortAlphabetic :: [String] -> [String]
sortAlphabetic xs = map toString $ sort $ map SortStr xs

parseInput :: [[Char]] -> (Either String [Option], [FilePath])
parseInput [] = (Right [], [])
parseInput (x:xs) = case x of
    [] -> parseInput xs
    (c:cs) -> case c of
        '-' -> let opts              = parseOpts cs
                   (rest_opts, file) = parseInput xs
               in ((do
                   os <- opts
                   os2 <- rest_opts
                   return (nub (os ++ os2)))
                   , file)
        _   -> (Right [], x:xs)

parseOpts :: [Char] -> Either String [Option]
parseOpts []     = Left "Empty option list provided following '-' token."
parseOpts (c:cs) = sequence $ go (c:cs)
    where go :: [Char] -> [Either String Option]
          go []          = []
          go (c:cs) =
              let opt = case c of
                            'r' -> return Reverse
                            'n' -> return Numeric
                            'u' -> return Unique
                            'c' -> return Compare
                            _   -> Left $ "Unknown option flag '" ++ c : "' provided in option list."
              in opt : go cs


sort :: (Ord a) => [a] -> [a]
sort []       = []
sort [x]      = [x]
sort (x:y:xs) = let (t:ts) = if x <= y
                             then x : sort (y:xs)
                             else y : sort (x:xs)
                in propogate t ts
                where
                    propogate :: (Ord b) => b -> [b] -> [b]
                    propogate t []     = [t]
                    propogate t (v:vs) = if t <= v then t : v : vs else v : (propogate t vs)
```

Usage

```
> ghc sort.hs
[1 of 1] Compiling Main             ( sort.hs, sort.o )
Linking sort ...

> cat test1.txt
10.5 hello
20.6 goodbye
13 Hello
13 Hello
0 Adios
19.2 Goodbye
23.9 Hola
0 Adios

> ./sort -n test1.txt
0 Adios
0 Adios
10.5 hello
13 Hello
13 Hello
19.2 Goodbye
20.6 goodbye
23.9 Hola

> ./sort -nu test1.txt
0 Adios
10.5 hello
13 Hello
19.2 Goodbye
20.6 goodbye
23.9 Hola

> ./sort -c
sort: disorder at line 1: 10.5 hello

> ./sort -n -r test1.txt
23.9 Hola
20.6 goodbye
19.2 Goodbye
13 Hello
13 Hello
10.5 hello
0 Adios
0 Adios

> ./sort -rnu test1.txt
23.9 Hola
20.6 goodbye
19.2 Goodbye
13 Hello
10.5 hello
0 Adios

> cat test2.txt
the
quick
brown
fox
jumped
over
the
lazy
dog

> ./sort test2.txt
brown
```

```
dog
fox
jumped
lazy
over
quick
the
the

> ./sort test1.txt test2.txt
0 Adios
0 Adios
10.5 hello
13 Hello
13 Hello
19.2 Goodbye
20.6 goodbye
23.9 Hola
brown
dog
fox
jumped
lazy
over
quick
the
the

> cat test3.txt
The
the
Quick
quick
Brown
brown
Fox
fox
Jumped
jumped
Over
over
The
the
Lazy
lazy
Dog

> ./sort test3.txt
brown
Brown
dog
Dog
fox
Fox
jumped
Jumped
lazy
Lazy
over
Over
quick
Quick
the
the
The
The

> ./sort -u test3.txt
brown
Brown
dog
Dog
```

```
fox
Fox
jumped
Jumped
lazy
Lazy
over
Over
quick
Quick
the
The

> ./sort -u test3.txt
The
the
Quick
quick
Over
over
Lazy
lazy
Jumped
jumped
Fox
fox
Dog
dog
Brown
brown

> cat test1.txt | ./sort -n
0 Adios
0 Adios
10.5 hello
13 Hello
13 Hello
19.2 Goodbye
20.6 goodbye
23.9 Hola

> cat test1.txt | ./sort -nur
23.9 Hola
20.6 goodbye
19.2 Goodbye
13 Hello
10.5 hello
0 Adios

> ./sort test1.txt test2.txt test3.txt
0 Adios
0 Adios
10.5 hello
13 Hello
13 Hello
19.2 Goodbye
20.6 goodbye
23.9 Hola
brown
brown
Brown
dog
dog
Dog
fox
fox
Fox
jumped
jumped
Jumped
lazy
lazy
```

```
Lazy
over
over
Over
quick
quick
Quick
the
the
the
the
The
The
```